```
// Computer Program Listing Appendix Under 37 CFR 1.52(e)
// dsre.c
// Copyright (c) 2004. Sybase, Inc.  All Rights Reserved.
/*
** Confidential property of Sybase, Inc.
*/
/*
**  This file has the database SRE routines:
**
** External Functions:
** --------------------
** dsre_build_dsre
** dsre_rebuild_dsre
** dsre_add_TnFnXnD_site
** dsre_remove_TnFnXnD_site
** dsre_resolve
**
** Internal Functions:
**      --------------------
** _dsre_build_TnFnXnD
** _dsre_init_dsre
** _dsre_add_TnFnXnD_entry
** ll_k_ins_ascent
*/
#include <dsre.h>
#define NUM_NSI_TABLES  4
#define NUM_HTS_TABLES  4
#define NUM_DSRE_BITMAPS 5
#define DSRE_NSI_TABLE_SIZE_C 7
STATIC_FUNC VOID_FUNC _dsre_init_dsre PROTOTYPE((
    DIST_INFO  *dist_info,
    RSHANDLE  *__RSHANDLE));
STATIC_FUNC VOID_FUNC _dsre_build_TnFnXnD  PROTOTYPE((
    DIST_INFO  *dist_info,
    LL_HDR   *hdr,
    RSHANDLE  *__RSHANDLE));
STATIC_FUNC VOID_FUNC _dsre_add_TnFnXnD_entry PROTOTYPE((
    DSRE_INFO  *dsre,
    CS_INT   status,
    OBJ_DBSUBSETS  *nameset,
    CS_INT   len,
    SITEID   dbid,
    CS_BOOL   truncate,
    RSHANDLE  *__RSHANDLE));
STATIC_FUNC VOID_FUNC ll_k_ins_ascent  PROTOTYPE((
    LL_HDR  *hdr,
    VOID  *item,
    LL_K_ELE *ele,
    CS_INT  len,
    VOID  *key));
```

```
/*
**  _DSRE_INIT_DSRE
**
**  Type of function: Internal.
**
**  Purpose:
** Allocate memory for the database SRE of a distributor
**
**  Parameters:
** DIST_INFO *dist_info (input)
**  A structure containing all information about the DIST
**  thread.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre becomes an active database resolution engine.
**
*/
STATIC_FUNC VOID_FUNC
_dsre_init_dsre(dist_info, __RSHANDLE)
DIST_INFO *dist_info;
RSHANDLE *__RSHANDLE;
{
 CS_INT  num_bits;
 MEM_EXPAND_HDR *memhdr;
 CS_INT  syncInfo;
 if (dist_info->dist_dsre != NULL)
  return;
 memhdr = (MEM_EXPAND_HDR*)NULL;
 /* Allocate memory for dist_dsre structure. */
 mem_allocate(&memhdr,
  (BYTE**) &dist_info->dist_dsre,
  sizeof(DSRE_INFO),
  MEM_EXPANSION_C, __RSHANDLE);
 MEMZERO(dist_info->dist_dsre, sizeof(DSRE_INFO));
 dist_info->dist_dsre->dsre_memhdr = memhdr;
 mem_unlink_hdr(dist_info->dist_dsre->dsre_memhdr, __RSHANDLE);
 dist_info->dist_dsre->dsre_tn = hts_create_nested("TABLE NSI",
  DSRE_NSI_TABLE_SIZE_C, hts_h_char1, NULL, __RSHANDLE);
 dist_info->dist_dsre->dsre_fn = hts_create_nested("FUNCTION NSI",
  DSRE_NSI_TABLE_SIZE_C, hts_h_char1, NULL, __RSHANDLE);
 dist_info->dist_dsre->dsre_xn = hts_create_nested("TRANSACTION NSI",
  DSRE_NSI_TABLE_SIZE_C, hts_h_char1, NULL, __RSHANDLE);
 dist_info->dist_dsre->dsre_spn = hts_create_nested("SYSSP NSI",
  DSRE_NSI_TABLE_SIZE_C, hts_h_char1, NULL, __RSHANDLE);
```

```
 mem_unlink_hdr(dist_info->dist_dsre->dsre_tn->memhdr, __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_fn->memhdr, __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_xn->memhdr, __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_spn->memhdr, __RSHANDLE);
 /*
 ** Allocate the bitmaps for dsre_*_default.
 ** These bitmaps are allocated once and kept used for
 ** life span of the dsre.
 */
 num_bits = sm_get_num_sites(SM_GLOBAL_INFO, __RSHANDLE);
 bm_get_bm(num_bits, num_bits,
   &dist_info->dist_dsre->dsre_tn_default, __RSHANDLE);
 bm_get_bm(num_bits, num_bits,
   &dist_info->dist_dsre->dsre_fn_default, __RSHANDLE);
 bm_get_bm(num_bits, num_bits,
   &dist_info->dist_dsre->dsre_xn_default, __RSHANDLE);
 bm_get_bm(num_bits, num_bits,
   &dist_info->dist_dsre->dsre_spn_default, __RSHANDLE);
 bm_get_bm(num_bits, num_bits,
   &dist_info->dist_dsre->dsre_subsites, __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_tn_default->bm_memhdr,
             __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_fn_default->bm_memhdr,
             __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_xn_default->bm_memhdr,
             __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_spn_default->bm_memhdr,
             __RSHANDLE);
 mem_unlink_hdr(dist_info->dist_dsre->dsre_subsites->bm_memhdr,
             __RSHANDLE);
 mem_transfer_hdr(dist_info->dist_dsre->dsre_memhdr,
  &((RSHANDLE_INFO*)G->g_rshandle_info)->rsinfo_global_rshandle.rs_exc_chain);
}
/*
**  DSRE_BUILD_DSRE
**
**  Type of function: External.
**
**  Purpose:
** Builds the database SRE for a distributor
**
**  Parameters:
** DIST_INFO *dist_info (input)
**  A structure containing all information about the DIST
**  thread.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
```

```
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre becomes an active database resolution engine.
**
*/
VOID_FUNC
dsre_build_dsre(dist_info, __RSHANDLE)
DIST_INFO *dist_info;
RSHANDLE *__RSHANDLE;
{
 STS_HANDLE  *stsh;
 OBJ_RSSUBSCRIPTIONS subrow;
 CS_CHAR   where[MAX_STRING_LEN_C + 1];
 LL_HDR   sub_lst;
 MEM_EXPAND_HDR  *dsre_memhdr;
 LL_K_SITEID  *siteid;
 stsh = sts_begin_trans("Build_DSRE", __RSHANDLE);
 SPRINTF(where, "pdbid = %ld and primary_sre = 1 and (type & %d) = %d",
  dist_info->dist_l_siteid, SUB_TYPE_DBSUB_M, SUB_TYPE_DBSUB_M);
 if (sts_exec_select(stsh, STS_SELECTOBJ_C,
  STS_TABLENAME(TYPE_RSSUBSCRIPTIONS_C), where,
  TYPE_RSSUBSCRIPTIONS_C, NULL, 0, __RSHANDLE) == CS_END_DATA)
 {
  stsh = sts_end_trans(stsh, STS_COMMIT_C, __RSHANDLE);
  return;
 }
 ll_init(&sub_lst);
 dsre_memhdr = (MEM_EXPAND_HDR*)NULL;
 while (sts_get_row(stsh, &subrow, TYPE_RSSUBSCRIPTIONS_C, __RSHANDLE)
  != CS_END_DATA)
 {
  mem_allocate(&dsre_memhdr, (BYTE**) &siteid,
   sizeof(LL_K_SITEID), MEM_EXPANSION_C, __RSHANDLE);
  SITEID_COPY(siteid->dbid, subrow.obj_sub.sub_dbid);
  RSID_COPY(siteid->repid, subrow.obj_sub.sub_objid);
  if (subrow.obj_sub.sub_status & SUB_STAT_ALLOW_TRUNC_M)
   siteid->allow_truncate = CS_TRUE;
  else
   siteid->allow_truncate = CS_FALSE;
  /* The sub_lst is an ascent list sorted by repid. */
  ll_k_ins_ascent(&sub_lst, siteid, &siteid->ele, sizeof(RSID),
      &siteid->repid);
 }
 stsh = sts_end_trans(stsh, STS_COMMIT_C, __RSHANDLE);
 if (dsre_memhdr != NULL)
 {
  _dsre_build_TnFnXnD(dist_info, &sub_lst, __RSHANDLE);
  ll_init(&sub_lst);
  mem_free(dsre_memhdr, __RSHANDLE);
```

```
 }
}
/*
**  DSRE_ADD_TnFnXnD_SITE
**
**  Type of function: External.
**
**  Purpose:
** This is the external version of _dsre_build_TnFnXnD function.
** unlike _dsre_build_TnFnXnD, this function just processes one database
** subscription.
**
**  Parameters:
** DIST_INFO *dist_info (input/output)
**  Pointer to the Distributor structure.
** SITEID  pdbid (input)
**  The primary dbid which is the source database of the dbrep.
**  We will use this dbid to locate the dist_info and its dsre.
** SITEID  dbid (input)
**  The site which newly subscribes to the database repdef.
** RSID  *dbrepid (input)
**  Pointer to the dbrepid of the database repdef.
** CS_BOOL  truncate (input)
**  Does this site subscribe to truncate table.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre grows by adding namesets for the new dbsub.
**
*/
VOID_FUNC
dsre_add_TnFnXnD_site(dist_info, pdbid, dbid, dbrepid, truncate, __RSHANDLE)
DIST_INFO *dist_info;
SITEID  pdbid;
SITEID  dbid;
RSID  *dbrepid;
CS_BOOL  truncate;
RSHANDLE *__RSHANDLE;
{
 LL_K_SITEID siteid;
 LL_HDR  hdr;
 SITEID_COPY(siteid.dbid, dbid);
 RSID_COPY(siteid.repid, *dbrepid);
 if (truncate)
    siteid.allow_truncate = CS_TRUE;
```

```
 else
    siteid.allow_truncate = CS_FALSE;
 ll_init(&hdr);
 /* For consistency, we still use ll_k_ins_ascent(), even though
 ** we can also use ll_k_insert().
 */
 ll_k_ins_ascent(&hdr, &siteid, &siteid.ele,
   sizeof(RSID), &siteid.repid);
 _dsre_build_TnFnXnD(dist_info, &hdr, __RSHANDLE);
}
/*
**  DSRE_REMOVE_TnFnXnD_SITE
**
**  Type of function: External.
**
**  Purpose:
** This routine will remove a site from the DSRE.
**
**  Parameters:
** DIST_INFO *dist_info (input)
**  Pointer to the Distributor structure. If this is NULL, use
**  pdbid to find it out.
** SITEID  pdbid (input)
**  The primary dbid which is the source database of the dbrep.
**  We will use this dbid to locate the dist_info and its dsre.
** SITEID  dbid (input)
**  The site which newly drop subscription to the database repdef.
** RSID  *dbrepid (input)
**  Pointer to the dbrepid of the database repdef.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre shrink by removing the site from the related
** namesets.
**
*/
VOID_FUNC
dsre_remove_TnFnXnD_site(dist_info, pdbid, dbid, __RSHANDLE)
DIST_INFO *dist_info;
SITEID  pdbid;
SITEID  dbid;
RSHANDLE *__RSHANDLE;
{
 DSRE_INFO *dsre;
 HTS_TABLE *tmp_table;
```

```
HTS_TABLE *hashtables[NUM_NSI_TABLES];
BM_STRUCT *defaultbms[NUM_NSI_TABLES];
SM_SITE_ID site_index;
CS_INT  index, hts_index;
CS_INT  syncInfo;       /* Error info from the Sync module */
if (dist_info == NULL)
{
    /* We need to find dist_info for this pdbid. */
    dist_info = ll_first(&DIST_G->distg_list);
    while (dist_info != (DIST_INFO *)NULL)
    {
 if (dist_info->dist_l_siteid == pdbid)
  break;
 else
  dist_info = ll_next(&DIST_G->distg_list,
   &dist_info->dist_next);
    }
}
if (dist_info == NULL)
{
 return;
}
dsre = dist_info->dist_dsre;
hashtables[0] = dsre->dsre_tn;
hashtables[1] = dsre->dsre_fn;
hashtables[2] = dsre->dsre_xn;
hashtables[3] = dsre->dsre_spn;
defaultbms[0] = dsre->dsre_tn_default;
defaultbms[1] = dsre->dsre_fn_default;
defaultbms[2] = dsre->dsre_xn_default;
defaultbms[3] = dsre->dsre_spn_default;
site_index = sm_get_index(SM_GLOBAL_INFO, dbid, __RSHANDLE);
bm_reset_bit(dsre->dsre_subsites, site_index);
if (bm_is_zero_bm(dsre->dsre_subsites))
{
    /* This is the last site subscribing to us. After this
    ** is dropped, there is no more database subscription
    ** for this primary site. We can drop the DSRE structure.
    */
    dsre_release_dsre(dist_info->dist_dsre, __RSHANDLE);
    dist_info->dist_dsre = NULL;
    return;
}
for (index = 0; index < NUM_NSI_TABLES; index++)
{
 tmp_table = hashtables[index];
 bm_reset_bit(defaultbms[index], site_index);
 for (hts_index = 0; hts_index < tmp_table->num_slots;
  hts_index++)
 {
```

```
   LL_HDR  *hdr;
   LL_K_ELE_2 *p;
   BM_STRUCT *tmp_bm;
   /* Traverse the whole list. */
   hdr = &tmp_table->table[hts_index];
   p = (LL_K_ELE_2 *)hdr->head;
   while (p != (LL_K_ELE_2 *) hdr)
   {
      tmp_bm = ((DSRE_ENTRY*)p->link.item)->dsre_e_bitmap;
      bm_reset_bit(tmp_bm, site_index);
      if (bm_is_zero_bm(tmp_bm))
      {
         LL_K_ELE_2 *tmp_ele;
    DSRE_ENTRY *tmp_entry;
         /* No site subscribes to this nameset,
         ** should remove it.
         */
         tmp_ele = p;
    p = (LL_K_ELE_2 *) p->link.next;
    tmp_entry = (DSRE_ENTRY*)tmp_ele->link.item;
         ll_del(&tmp_ele->link);
    mem_free(tmp_entry->dsre_e_bitmap->bm_memhdr, __RSHANDLE);
    mem_free(tmp_entry->dsre_e_memhdr, __RSHANDLE);
      }
      else
      {
    p = (LL_K_ELE_2 *) p->link.next;
      }
   }
  }
 }
}
/*
**  DSRE_RESOLVE
**
**  Type of function: External.
**
**  Purpose:
** This routine will return a bitmap for sites based on the input
** name pair.
**
**  Parameters:
** DSRE_INFO *dsre (input)
**  A structure containing hash tables needed to resolve
**  the name pair.
** CS_INT  type (input)
**  Where this name pair is for table, function or transaction.
** CS_CHAR  *owner (input)
**  The first key of the name pair
** CS_CHAR  *name (input)
```

```
**  The second key of the name pair
** BM_STRUCT *dests (output)
**  The bitmap structure to hold the result.
**  For transaction resolution, we allow this to be NULL.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
**
*/
VOID_FUNC
dsre_resolve(dsre, type, owner, name, dests, __RSHANDLE)
DSRE_INFO *dsre;
CS_INT  type;
CS_CHAR  *owner;
CS_CHAR  *name;
BM_STRUCT *dests;
RSHANDLE *__RSHANDLE;
{
 HTS_TABLE *nameset;
 BM_STRUCT *def;
 DSRE_ENTRY *tmp_entry;
 CS_CHAR  key1[MAX_IDENT_LEN_C+1];
 CS_CHAR  key2[MAX_IDENT_LEN_C+1];
 CS_INT  len1, len2, len3;
 CS_BOOL  retry_key1;
 CS_BOOL  resolved;
 CS_BOOL  found_regular_sub;
 CS_INT  num_bits;
 CS_INT  syncInfo;
 if (dsre == NULL)
  return;
 num_bits = sm_get_num_sites(SM_GLOBAL_INFO, __RSHANDLE);
 switch (type)
 {
    case DBSUBSET_TYPE_TABLE_C:
  nameset = dsre->dsre_tn;
  def = dsre->dsre_tn_default;
  break;
    case DBSUBSET_TYPE_FUNCTION_C:
  nameset = dsre->dsre_fn;
  def = dsre->dsre_fn_default;
  break;
    case DBSUBSET_TYPE_SYSSP_C:
  nameset = dsre->dsre_spn;
  def = dsre->dsre_spn_default;
```

```
 break;
    case DBSUBSET_TYPE_TRAN_C:
 nameset = dsre->dsre_xn;
 def = dsre->dsre_xn_default;
 break;
}
bm_reset_bm(dests);
if (owner == NULL || owner[0] == NULLCHAR)
 STRCPY(key1, DBSUBSET_NONAME_S);
else
 STRCPY(key1, owner);
if (STRCMP(key1, DBSUBSET_WILDCAST_S) == 0)
{
 retry_key1 = CS_FALSE;
}
else
{
 retry_key1 = CS_TRUE;
}
if (name == NULL || name[0] == NULLCHAR)
 STRCPY(key2, DBSUBSET_NONAME_S);
else
 STRCPY(key2, name);
len1 = STRLEN(key1);
len2 = STRLEN(key2);
len3 = STRLEN(DBSUBSET_WILDCAST_S);
resolved = CS_FALSE;
if ((tmp_entry = (DSRE_ENTRY*)hts_find_2(nameset,
  len1, key1, len2, key2)) != NULL)
{
 /* Because we are not sure whether tmp_entry->dsre_e_bitmap
 ** has a shorter length or not, we need to copy it
 ** to a temporary bitmap in order to avoid bm_or_bms()
 ** failure. This is the same for the following codes.
 */
 bm_copy_bm(dests, tmp_entry->dsre_e_bitmap);
 resolved = CS_TRUE;
}
if (retry_key1)
{
 if ((tmp_entry = (DSRE_ENTRY*)hts_find_2(nameset,
  len3, DBSUBSET_WILDCAST_S,
  len2, key2)) != NULL)
 {
  bm_or_bms(dests, tmp_entry->dsre_e_bitmap, dests);
  resolved = CS_TRUE;
 }
 if ((tmp_entry = (DSRE_ENTRY*)hts_find_2(nameset,
  len1, key1, len3, DBSUBSET_WILDCAST_S)) != NULL)
 {
```

```
   bm_or_bms(dests, tmp_entry->dsre_e_bitmap, dests);
   resolved = CS_TRUE;
  }
 }
 if (! resolved)
 {
  bm_copy_bm(dests, def);
 }
}
/*
**  _DSRE_BUILD_TnFnXnD
**
**  Type of function: Internal.
**
**  Purpose:
** Add a list of siteid to a DSRE
**
**  Parameters:
** DIST_INFO *dist_info (input)
**  A structure containing all information about the DIST
**  thread.
** LL_HDR  *hdr (input)
**  A linked list holding the siteids to be added into the DSRE.
**  of the DIST thread.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre becomes an active database resolution engine.
**
*/
STATIC_FUNC VOID_FUNC
_dsre_build_TnFnXnD(dist_info, hdr, __RSHANDLE)
DIST_INFO *dist_info;
LL_HDR  *hdr;
RSHANDLE *__RSHANDLE;
{
 LL_K_SITEID *siteid;
 VOID  *sts_obj;
 CS_INT  status;
 STS_REPEAT_OBJ *repobj;
 LL_HDR  *lst;
 OBJ_DBSUBSETS *nameset;
 RSID  tmp_dbrepid;
 CS_INT  index;
 DSRE_INFO *dsre;
```

```
if (dist_info->dist_dsre == (DSRE_INFO*)NULL)
{
 _dsre_init_dsre(dist_info, __RSHANDLE);
}
dsre = dist_info->dist_dsre;
siteid = (LL_K_SITEID*)ll_first(hdr);
while (siteid != (LL_K_SITEID*)NULL)
{
 if (sts_get_obj(NULL, TYPE_RSDBREPS_KEY2_C, &siteid->repid,
  STS_KEY_STRUCT_C, (VOID**)&sts_obj, __RSHANDLE)
     != SUCCEED)
 {
  CS_CHAR objid_str[RSID_HEX_STRLEN + 1];
  RSID_TO_HEX_STR(siteid->repid, objid_str);
  /* Should not happen, raise exception anyway. */
  exc_raise_exception(EXC_RETRYABLE_M,
   ERR_MKERR(DDL, DDL_UNKNOWN_OBJ),
   TRUE, __LINE__, __FILE__,
   objid_str);
 }
 status = ((OBJ_RSDBREPS*)
  GET_SHARERESOURCE(sts_obj))->obj_dbrep.dbrep_status;
 sts_obj = sts_free_obj(sts_obj, __RSHANDLE);
 if ((status & DBREP_ST_TABLES1_M) ||
     (status & DBREP_ST_FUNCTIONS1_M) ||
     (status & DBREP_ST_TRANS1_M) ||
     (status & DBREP_ST_SYSSP1_M))
 {
  OBJ_RSDBSUBSETS *tmpset;
  if (sts_get_obj(NULL, TYPE_RSDBSUBSETS_KEY3_C,
   &siteid->repid, STS_KEY_STRUCT_C,
   (VOID**)&sts_obj, __RSHANDLE) != SUCCEED)
  {
      CS_CHAR objid_str[RSID_HEX_STRLEN + 1];
      RSID_TO_HEX_STR(siteid->repid, objid_str);
      /* Should not happen, raise exception anyway. */
      exc_raise_exception(EXC_RETRYABLE_M,
   ERR_MKERR(DDL, DDL_UNKNOWN_OBJ),
   TRUE, __LINE__, __FILE__,
   objid_str);
  }
  repobj = (STS_REPEAT_OBJ*) GET_SHARERESOURCE(sts_obj);
  lst = &repobj->repobj_llhdr;
  nameset = (OBJ_DBSUBSETS*)MALLOC(sizeof(OBJ_DBSUBSETS)
   *ll_list_len(lst));
  for (index=0, tmpset=(OBJ_RSDBSUBSETS*)ll_first(lst);
   tmpset != (OBJ_RSDBSUBSETS*)NULL;
   index++,
   tmpset = (OBJ_RSDBSUBSETS*)ll_next(lst,
     &tmpset->obj_basic.obj_linked_entry))
```

```
  {
   MEMCPY(&nameset[index], &tmpset->obj_dbsubset,
    sizeof(OBJ_DBSUBSETS));
  }
  sts_obj = sts_free_obj(sts_obj, __RSHANDLE);
 }
 else
 {
  nameset = (OBJ_DBSUBSETS*)NULL;
  index = 0;
 }
 _dsre_add_TnFnXnD_entry(dsre, status, nameset, index,
   siteid->dbid, siteid->allow_truncate,
                __RSHANDLE);
 RSID_COPY(tmp_dbrepid, siteid->repid);
 /* We have multiple subscription site for the same
 ** database repdef.
 */
 while ((siteid = (LL_K_SITEID*)ll_next(hdr,
  &siteid->ele.link)) != (LL_K_SITEID*)NULL &&
  RSID_CMP(siteid->repid, tmp_dbrepid) ==0)
 {
  _dsre_add_TnFnXnD_entry(dsre, status, nameset, index,
    siteid->dbid, siteid->allow_truncate,
                 __RSHANDLE);
 }
 if (nameset != (OBJ_DBSUBSETS*)NULL)
  FREE(nameset);
 }
}
/*
**  _DSRE_ADD_TnFnXnD_ENTRY
**
**  Type of function: Internal.
**
**  Purpose:
** Add a list of nameset with a dbid to a DSRE
**
**  Parameters:
** DSRE_INFO *dsre (input/output)
**  A structure to add the sites to.
** CS_INT  status (input)
**  The status of the related dbrep.
** OBJ_DBSUBSETS *namesets (input)
**  An array of namesets to be added for the input dbid.
** CS_INT  len (input)
**  Size of the namesets array.
** SITEID  dbid (input)
**  The dbid which those namesets will be added for.
** CS_BOOL  truncate (input)
```

```
**  Does this site subscribe to truncate table.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre get more item inserted into its hash tables.
**
*/
STATIC_FUNC VOID_FUNC
_dsre_add_TnFnXnD_entry(dsre, status, nameset, len, dbid, truncate, __RSHANDLE)
DSRE_INFO  *dsre;
CS_INT   status;
OBJ_DBSUBSETS  *nameset;
CS_INT   len;
SITEID   dbid;
CS_BOOL   truncate;
RSHANDLE  *__RSHANDLE;
{
 CS_INT  index, hts_index;
 HTS_TABLE *tmp_table;
 BM_STRUCT **tmp_def;
 CS_INT  st1, st2;
 SM_SITE_ID site_index;
 DSRE_ENTRY *tmp_entry;
 CS_CHAR  *key1, *key2;
 CS_INT  len1, len2;
 CS_CHAR  type;
 CS_INT  allst1[NUM_NSI_TABLES];
 CS_INT  allst2[NUM_NSI_TABLES];
 HTS_TABLE *alltables[NUM_NSI_TABLES];
 BM_STRUCT **alldefaults[NUM_NSI_TABLES];
 CS_INT  num_bits;
 MEM_EXPAND_HDR *memhdr;
 CS_INT  syncInfo;
 allst1[0] = DBREP_ST_TABLES1_M;
 allst1[1] = DBREP_ST_FUNCTIONS1_M;
 allst1[2] = DBREP_ST_TRANS1_M;
 allst1[3] = DBREP_ST_SYSSP1_M;
 allst2[0] = DBREP_ST_TABLES2_M;
 allst2[1] = DBREP_ST_FUNCTIONS2_M;
 allst2[2] = DBREP_ST_TRANS2_M;
 allst2[3] = DBREP_ST_SYSSP2_M;
 alltables[0] = dsre->dsre_tn;
 alltables[1] = dsre->dsre_fn;
 alltables[2] = dsre->dsre_xn;
 alltables[3] = dsre->dsre_spn;
```

```
alldefaults[0] = &(dsre->dsre_tn_default);
alldefaults[1] = &(dsre->dsre_fn_default);
alldefaults[2] = &(dsre->dsre_xn_default);
alldefaults[3] = &(dsre->dsre_spn_default);
site_index = sm_get_index(SM_GLOBAL_INFO, dbid, __RSHANDLE);
num_bits = sm_get_num_sites(SM_GLOBAL_INFO, __RSHANDLE);
bm_expand_bm(num_bits, BM_COPY_C, &(dsre->dsre_subsites),
 __RSHANDLE);
bm_set_bit(dsre->dsre_subsites, site_index);
for (index = 0; index < NUM_NSI_TABLES; index++)
{
 if (((status & allst1[index]) && (status & allst2[index])) ||
     (!(status & allst1[index]) && !(status & allst2[index])))
 {
  tmp_table = alltables[index];
  /*
  ** If the db repdef indicate
  ** (1) "not replicate {tables|functions|transactions|
  **       system procedures} in (...) "
  ** (2) "replicate {tables|functions|transactions|
  **       system procedures} (without in clause)
  **
  ** We need to update all elements in the hash table.
  */
  for (hts_index = 0; hts_index < tmp_table->num_slots;
   hts_index++)
  {
   LL_HDR  *hdr;
   LL_K_ELE_2 *p;
   hdr = &tmp_table->table[hts_index];
   for (p = (LL_K_ELE_2 *)hdr->head;
    p != (LL_K_ELE_2 *) hdr;
    p = (LL_K_ELE_2 *) p->link.next)
   {
       bm_expand_bm(num_bits,
           BM_COPY_C,
    &(((DSRE_ENTRY*)p->link.item)->dsre_e_bitmap),
           __RSHANDLE);
       bm_set_bit(((DSRE_ENTRY*)
     p->link.item)->dsre_e_bitmap,
     site_index);
   }
  }
 }
}
/* If nameset is NULL, len must be 0 and therefore we are not
** falling into the loop below.
*/
for (index = 0; index < len; index++)
{
```

```
type = nameset[index].dbsubset_type;
switch (type)
{
  case DBSUBSET_TYPE_TABLE_C:
 st1 = DBREP_ST_TABLES1_M;
 st2 = DBREP_ST_TABLES2_M;
 tmp_table = dsre->dsre_tn;
 tmp_def = &(dsre->dsre_tn_default);
 break;
  case DBSUBSET_TYPE_FUNCTION_C:
 st1 = DBREP_ST_FUNCTIONS1_M;
 st2 = DBREP_ST_FUNCTIONS2_M;
 tmp_table = dsre->dsre_fn;
 tmp_def = &(dsre->dsre_fn_default);
 break;
  case DBSUBSET_TYPE_TRAN_C:
 st1 = DBREP_ST_TRANS1_M;
 st2 = DBREP_ST_TRANS2_M;
 tmp_table = dsre->dsre_xn;
 tmp_def = &(dsre->dsre_xn_default);
 break;
  case DBSUBSET_TYPE_SYSSP_C:
 st1 = DBREP_ST_SYSSP1_M;
 st2 = DBREP_ST_SYSSP2_M;
 tmp_table = dsre->dsre_spn;
 tmp_def = &(dsre->dsre_spn_default);
 break;
  default:
 trc_print("Unknown subset type: %d\n", type);
 break;
}
key1 = nameset[index].dbsubset_owner;
key2 = nameset[index].dbsubset_name;
len1 = STRLEN(key1);
len2 = STRLEN(key2);
if ((tmp_entry = (DSRE_ENTRY*)hts_find_2(
 tmp_table, len1, key1, len2, key2)) == NULL)
{
 memhdr = (MEM_EXPAND_HDR*)NULL;
 mem_allocate(&memhdr,
  (BYTE**)&tmp_entry,
  sizeof(DSRE_ENTRY),
  MEM_EXPANSION_C, __RSHANDLE);
 tmp_entry->dsre_e_memhdr = memhdr;
 mem_unlink_hdr(tmp_entry->dsre_e_memhdr, __RSHANDLE);
 STRCPY(tmp_entry->dsre_e_owner, key1);
 STRCPY(tmp_entry->dsre_e_name, key2);
 bm_get_bm(sm_get_num_sites(SM_GLOBAL_INFO, __RSHANDLE),
    sm_get_num_sites(SM_GLOBAL_INFO, __RSHANDLE),
    &(tmp_entry->dsre_e_bitmap), __RSHANDLE);
```

```
  mem_unlink_hdr(tmp_entry->dsre_e_bitmap->bm_memhdr, __RSHANDLE);
  bm_copy_bm(tmp_entry->dsre_e_bitmap, *tmp_def);
  hts_insert_2(tmp_table, tmp_entry,
   &tmp_entry->dsre_e_h_entry,
   len1, tmp_entry->dsre_e_owner,
   len2, tmp_entry->dsre_e_name);
 }
 if (!(status & st2))
 {
  /* Not negated. */
  bm_set_bit(tmp_entry->dsre_e_bitmap, site_index);
 }
 else
 {
  /* Negate. */
  bm_reset_bit(tmp_entry->dsre_e_bitmap, site_index);
 }
}
for (index = 0; index < NUM_NSI_TABLES; index++)
{
 if (((status & allst1[index]) && (status & allst2[index])) ||
     (!(status & allst1[index]) && !(status & allst2[index])))
 {
  /*
  ** If the db repdef indicate
  ** (1) "not replicate {tables|functions|transactions|
  **      system procedures} in (...) "
  ** (2) "replicate {tables|functions|transactions|
  **      system procedures} (without in clause)
  **
  ** We need to store the this site in the related
  ** default bitmap.
  */
  bm_expand_bm(num_bits,
        BM_COPY_C, alldefaults[index], __RSHANDLE);
  bm_set_bit(*alldefaults[index], site_index);
 }
 }
}
/*
**  LL_K_INS_ASCENT
**
**  Type of function: Internal.
**
**  Purpose:
** Add an item into a list with ascendent order.
**
**  Parameters:
** LL_HDR  *hdr (input)
**  A linked list the new item being inserted into.
```

```
** VOID  *item (input)
**  A data item being inserted into the list. The element is part
**  of the data.
** LL_K_ELE *ele (input-output)
**  Keyed Linked list element that is added to the head of the list.
** CS_INT  len (input)
**  Length of the key
** VOID  *key (input)
**  Pointer to the key, this should also be part of the data item.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** hdr grows with an item inserted ascendently.
**
*/
STATIC VOID_FUNC
ll_k_ins_ascent(hdr, item, ele, len, key)
LL_HDR  *hdr;
VOID  *item;
LL_K_ELE *ele;
CS_INT  len;
VOID  *key;
{
 LL_K_ELE *tmp_ele;
 ele->len = len;
 ele->key = key;
 /* The hdr is an ascent list sorted by key. */
 if ((LL_K_ELE*)hdr->head == (LL_K_ELE*)hdr)
 {
  ll_ins_head(hdr, item, &ele->link);
 }
 else
 {
  tmp_ele = (LL_K_ELE *) hdr->head;
  while ((tmp_ele != (LL_K_ELE *) hdr) &&
         (ll_keycmp(key, tmp_ele->key, MIN(len, tmp_ele->len),
    hdr->strcmp_info, 0) < 0))
  {
   tmp_ele = (LL_K_ELE *) tmp_ele->link.next;
  }
  if (tmp_ele != (LL_K_ELE *) hdr)
  {
   ll_ins_before(hdr, &tmp_ele->link,
    item, &ele->link);
  }
  else
  {
```

```
   ll_ins_tail(hdr, item, &ele->link);
  }
 }
}
/*
**  DSRE_RELEASE_DSRE
**
**  Type of function: Internal.
**
**  Purpose:
** To release the memory ocupied by the DSRE_INFO structure.
**
**  Parameters:
** DSRE_INFO *dsre (input/output)
**  The structure to be released.
**
**  Returns:
** VOID.
**
**  Side Effects:
**
*/
VOID_FUNC
dsre_release_dsre(dsre, __RSHANDLE)
DSRE_INFO *dsre;
RSHANDLE *__RSHANDLE;
{
 HTS_TABLE *tmp_table;
 HTS_TABLE *hashtables[NUM_HTS_TABLES];
 BM_STRUCT *dsre_bms[NUM_DSRE_BITMAPS];
 CS_INT  index, hts_index;
 hashtables[0] = dsre->dsre_tn;
 hashtables[1] = dsre->dsre_fn;
 hashtables[2] = dsre->dsre_xn;
 hashtables[3] = dsre->dsre_spn;
 dsre_bms[0] = dsre->dsre_tn_default;
 dsre_bms[1] = dsre->dsre_fn_default;
 dsre_bms[2] = dsre->dsre_xn_default;
 dsre_bms[3] = dsre->dsre_spn_default;
 dsre_bms[4] = dsre->dsre_subsites;
 for (index = 0; index < NUM_HTS_TABLES; index++)
 {
  tmp_table = hashtables[index];
  for (hts_index = 0; hts_index < tmp_table->num_slots;
   hts_index++)
  {
   LL_HDR  *hdr;
   LL_K_ELE_2 *p;
   DSRE_ENTRY *tmp_entry;
   /* Traverse and release the whole list. */
```

```
   hdr = &tmp_table->table[hts_index];
   p = (LL_K_ELE_2 *)hdr->head;
   while (p != (LL_K_ELE_2 *) hdr)
   {
    tmp_entry = (DSRE_ENTRY*)p->link.item;
    ll_del(&p->link);
    mem_free(tmp_entry->dsre_e_bitmap->bm_memhdr, __RSHANDLE);
    mem_free(tmp_entry->dsre_e_memhdr, __RSHANDLE);
    p = (LL_K_ELE_2 *)hdr->head;
   }
  }
  mem_free(tmp_table->memhdr, __RSHANDLE);
 }
 for (index = 0; index < NUM_DSRE_BITMAPS; index++)
 {
  mem_free(dsre_bms[index]->bm_memhdr, __RSHANDLE);
 }
 mem_free(dsre->dsre_memhdr, __RSHANDLE);
}
/*
**  DSRE_REBUILD_DSRE
**
**  Type of function: External.
**
**  Purpose:
** Rebuilds the database SRE for a distributor and a particular db repdef.
**
**  Parameters:
** DIST_INFO *dist_info (input)
**  A structure containing all information about the DIST
**  thread.
** RSID  *dbrepid (input)
**  Pointer to the db repdef ID.
** RSHANDLE *__RSHANDLE
**  Pointer to thread specific info.
**
**  Returns:
** VOID.
** A exception is raised if an error occurs.
**
**  Side Effects:
** dist_info->dist_dsre is resynched.
**
*/
VOID_FUNC
dsre_rebuild_dsre(dist_info, dbrepid, __RSHANDLE)
DIST_INFO *dist_info;
RSID  *dbrepid;
RSHANDLE *__RSHANDLE;
{
```

```
 OBJ_RSSUBSCRIPTIONS *subrow;
 LL_HDR   sub_lst;
 MEM_EXPAND_HDR  *dsre_memhdr;
 LL_K_SITEID  *siteid;
 KEY15_SUBSCRIPTIONS sub_key;
 VOID   *sts_obj;
 STS_REPEAT_OBJ  *repobj;
 LL_HDR   *rep_llhdr;
 /* No subscriptions yet, do nothing. */
 if (dist_info->dist_dsre == NULL)
    return;
 RSID_COPY(sub_key.sub_objid, *dbrepid);
 sub_key.sub_primary_sre = 1;
 if (sts_get_obj((STS_HANDLE*)NULL, TYPE_RSSUBSCRIPTIONS_KEY15_C,
  &sub_key, STS_KEY_STRUCT_C, &sts_obj, __RSHANDLE) == SUCCEED)
 {
  repobj = (STS_REPEAT_OBJ*)GET_SHARERESOURCE(sts_obj);
  rep_llhdr = &repobj->repobj_llhdr;
  ll_init(&sub_lst);
  dsre_memhdr = (MEM_EXPAND_HDR*)NULL;
  for (subrow = (OBJ_RSSUBSCRIPTIONS*)ll_first(rep_llhdr);
      subrow != (OBJ_RSSUBSCRIPTIONS*)NULL;
      subrow = (OBJ_RSSUBSCRIPTIONS*)ll_next(rep_llhdr,
                 &subrow->obj_basic.obj_linked_entry))
  {
   mem_allocate(&dsre_memhdr, (BYTE**) &siteid,
      sizeof(LL_K_SITEID), MEM_EXPANSION_C, __RSHANDLE);
   SITEID_COPY(siteid->dbid, subrow->obj_sub.sub_dbid);
   RSID_COPY(siteid->repid, subrow->obj_sub.sub_objid);
   if (subrow->obj_sub.sub_status & SUB_STAT_ALLOW_TRUNC_M)
    siteid->allow_truncate = CS_TRUE;
   else
    siteid->allow_truncate = CS_FALSE;
   /* The sub_lst is an ascent list sorted by repid. */
   ll_k_ins_ascent(&sub_lst, siteid, &siteid->ele,
              sizeof(RSID), &siteid->repid);
   /* We drop all subscribing site for this db repdef. */
   dsre_remove_TnFnXnD_site(dist_info,
                       subrow->obj_sub.sub_pdbid,
                       siteid->dbid, __RSHANDLE);
  }
  sts_obj = sts_free_obj(sts_obj, __RSHANDLE);
 _dsre_build_TnFnXnD(dist_info, &sub_lst, __RSHANDLE);
 ll_init(&sub_lst);
 mem_free(dsre_memhdr, __RSHANDLE);
 }
}
// dsre.h
// Copyright (c) 2004. Sybase, Inc.  All Rights Reserved.
/*
```

```
** Confidential property of Sybase, Inc.
*/
/*
** generic/include/dsre.h:
**
**  This file defines the database SRE data structures.
*/
#ifndef __dsre__
#define __dsre__
#include <central.h>
#include <err.h>
#include <cmd.h>
#include <td.h>
typedef struct _dsre_entry
{
 MEM_EXPAND_HDR *dsre_e_memhdr; /* memory for this structure */
 HTS_ENTRY_2 dsre_e_h_entry; /* handle in h_table. */
 CS_CHAR  dsre_e_owner[MAX_IDENT_LEN_C+1];
 CS_CHAR  dsre_e_name[MAX_IDENT_LEN_C+1];
 BM_STRUCT *dsre_e_bitmap; /* sites subscribe to this name */
} DSRE_ENTRY;
typedef struct _dsre_info
{
 MEM_EXPAND_HDR *dsre_memhdr; /* memory for this structure */
 HTS_TABLE *dsre_tn; /* Table NSI. */
 HTS_TABLE *dsre_fn; /* Function NSI. */
 HTS_TABLE *dsre_xn; /* Transaction NSI. */
 HTS_TABLE *dsre_spn; /* System Procedure NSI. */
 BM_STRUCT *dsre_tn_default;
    /* Default subscription sites for tn */
 BM_STRUCT *dsre_fn_default;
    /* Default subscription sites for fn */
 BM_STRUCT *dsre_xn_default;
    /* Default subscription sites for xn */
 BM_STRUCT *dsre_spn_default;
    /* Default subscription sites for spn */
 BM_STRUCT *dsre_subsites; /* All sites subscribe to us. */
} DSRE_INFO;
typedef struct _ll_k_siteid
{
 SITEID  dbid;
 RSID  repid;
 CS_BOOL  allow_truncate;
 CS_BOOL  is_article;
 LL_K_ELE ele;
} LL_K_SITEID;
VOID_FUNC dsre_resolve  PROTOTYPE((
   DSRE_INFO *dsre,
   CS_INT  type,
   CS_CHAR  *owner,
```

```
    CS_CHAR  *name,
    BM_STRUCT *dests,
    RSHANDLE *__RSHANDLE));
VOID_FUNC dsre_release_dsre PROTOTYPE((
    DSRE_INFO *dsre,
    RSHANDLE *__RSHANDLE));
#endif /* __dsre__ */
```